|  |  |  |  |
| --- | --- | --- | --- |
|  | Python | JS | Php |
| Output | print()  print(a,b)  print(a+b) | console.log()  console.log(a,b)  console.log(a+b) | echo, print(), print\_r()  echo $a,$b  echo $a.$b |
| Input | username = input("Enter username: ") | const readline=require('readline').createInterface({ input: process.stdin, output: process.stdout })  readline.question(`What's your name?`, name => {  console.log(`Hi ${name}!`)  readline.close()  }) | $x=readline(‘what is your answer?’); |
| Exit | quit() | process.exit() | die() |
| Export | Normal saved py | module.exports=testapp; //nodejs default export  module.exports={ testapp }; //nodejs named exp  export default name  export name | Normal saved php files |
| Import | import module //module.py  import module as mx  from module import person1  Usage  module.var  module.func()  module.Class()  Print(dir(module)) //print all methods on module | const module = require('')  const module, { module, module, ... } = require('')  import name from ''  import name as newname from ''  import {name} from ''  import \* as name from ‘’  Usage  Module  module.prop  module()  new module.Class() | require()  require\_once  include()  include\_once()  Usage  Access all vars, funcs, and classes as usual |
| Casting | int(x), str(x), bool(x), float(x.xx) | Number(x), String(x), Boolean(x) | (int)1, (string)’str’, (array), (bool), (float), (object), (mixed) |
| Concatenation | a+b | a+b | a.b |
| Strings | ‘’, ””, ‘’’ ‘’’, “”” “””  ‘string’. methodname() | ‘’, ””,  ‘string’. charAt()  charCodeAt()  codePointAt()  concat()  constructor  endsWith()  fromCharCode()  includes()  indexOf()  lastIndexOf()  length  localeCompare()  match()  prototype  repeat()  replace()  search()  slice()  split()  startsWith()  substr()  substring()  toLocaleLowerCase()  toLocaleUpperCase()  toLowerCase()  toString()  toUpperCase()  trim()  trimEnd()  trimStart()  valueOf() | ‘’, ””,  addcslashes(‘’)  addslashes()  bin2hex()  chunk\_split()  count\_chars()  echo()  explode()  hex2bin()  htmlentities()  htmlspecialchars()  implode()  join()  lcfirst()  ltrim()  md5()  md5\_file()  money\_format()  nl2br()  number\_format()  parse\_str()  print()  rtrim()  str\_getcsv()  str\_ireplace()  str\_pad()  str\_repeat()  str\_replace()  str\_rot13()  str\_shuffle()  str\_split()  str\_word\_count()  strcasecmp()  strcmp()  strip\_tags()  stripcslashes()  stripslashes()  stripos()  stristr()  strlen()  strpbrk()  strpos()  strrchr()  strrev()  strstr()  strtok()  strtolower()  strtoupper()  strtr()  substr()  substr\_compare()  substr\_count()  substr\_replace()  trim()  ucfirst()  ucwords()  wordwrap() |
| Array | ["apple", "banana", "cherry"] //list  () //tuple, {} //set  Methods: list.append()  clear()  copy()  count()  extend()  index()  insert()  pop()  remove()  reverse()  sort() | ["apple", "banana", "cherry"]  []. concat()  constructor  copyWithin()  entries()  every()  fill()  filter()  find()  findIndex()  forEach()  from()  includes()  indexOf()  isArray()  join()  keys()  lastIndexOf()  length  map()  pop()  prototype  push()  reduce()  reduceRight()  reverse()  shift()  slice()  some()  sort()  splice()  toString()  unshift()  valueOf() | array("Volvo", "BMW", "Toyota");  ["Volvo", "BMW", "Toyota"]  array()  array\_chunk()  array\_count\_values()  array\_diff()  array\_diff\_assoc()  array\_diff\_key()  array\_fill()  array\_filter()  array\_key\_exists()  array\_keys()  array\_map()  array\_merge()  array\_multisort()  array\_pad()  array\_pop()  array\_push()  array\_rand()  array\_reduce()  array\_reverse()  array\_search()  array\_shift()  array\_slice()  array\_splice()  array\_sum()  array\_unique()  array\_unshift()  array\_values()  arsort()  asort()  compact()  count()  each()  in\_array()  key()  krsort()  ksort()  list()  range()  rsort()  shuffle()  sort() |
| Object | { "brand": "Ford", "model": "Mustang"} //dict  Dict. clear()  copy()  fromkeys()  get()  items()  keys()  pop()  popitem()  setdefault()  update()  values() | { brand: "Ford", model: "Mustang"} | array("Peter"=>"35", "Ben"=>"37");  ["Peter"=>"35", "Ben"=>"37"] |
| Conditions | If (a>b):  elif:  else: | If(a>b){}  else if  else  switch(expression) {  case x:  break;  default:  } | If(a>b){}  elseif  else  switch(expression) {  case x:  break;  default:  } |
| Logical | and, or, not, is, is not, in, not in | &&, ||, ! | and &&, or ||, ! |
| Loops | i = 1  while i < 6:  i +=1  else:  for x in arr|obj|str:  else:  //arr x,obj x, obj[x], str[x] | for (var i = 0, str=””; i < arr.length; i++) {}  for (let x of arr) {x} //arr  for (let x in obj) {obj[x]} //obj  while (i < 10) { i++ }  do { i++ } while (i < 10); | for ($x = 0; $x <= 10; $x++) {}  foreach ($arr as $x) { } //arr  foreach($obj as $x => $val) { } //obj  while($x <= 5) { $x++ }  do { $x++ } while ($x <= 5); |
| Function | def my\_function(arg=”str”): arg  def my\_function(\*args):  args[0],args[1]  def my\_function(\*\*args):  args[‘name’],args[‘name’]  x = lambda a,b : a + b  my\_function(arg1 = "1", arg2 = "2", arg3 = "3")  return str, arr, obj  return lambda a: a+10 | function name(arg=”str”){ arg }  var name = function(…args){ }  var name = () => { }  var x = (x, y) => x \* y;  (function(){ })(); (()=>{ })()  name(arg1 = "1", arg2 = "2", arg3 = "3")  return str, arr, obj  return function | function name(int $arg=”str”): int { $arg }  $name = function(…$args){ }  (function(){ }); //test this  $name = fn($a) => $str . $a;  function name() use($var){ }  name($arg1 = "1", $arg2 = "2", $arg3 = "3")  return (string) str  types: array, bool, float, int, string, object, mixed. |
| Global | global x | Create var in main body | $GLOBALS[‘x’] |
|  | try:  except:  else: | finally:    raise Exception(‘’) | try {}  catch(err) {}  finally {}  throw 'err'; | try {}  catch (\Throwable $th) { //throw $th;}  finally{}  throw 'err'; |
| Class | class Person:  def \_\_init\_\_(this, name, age):  this.name = name  this.age = age  x=5  def myfunc(this):  print("Hello my name is " + this.name)  p1 = Person("John", 36)  print(p1.x)  print(p1.name)  print(p1.age)  p1.myfunc()  p1.age = 40  p1.x=15  //self can replace this, alwasy 1st arg in all funcs | class Person {  constructor(name, age){  this.name = name  this.age = age  }    var x=5  static hello() { console.log( "Hello!!") }  myfunc(){  console.log("Hello my name is " + this.name)  }  }  Person.hello()  var p1 = new Person("John", 36)  console.log (p1.x)  console.log (p1.name)  console.log (p1.age)  p1.myfunc()  p1.age = 40  p1.x=15  function person(name,age) {  this.name=str;  this.age=age;  var x= 5;  function hello(){ console.log('Hello')}  this.myfunc=function(){ console.log("Hello my name is " + this.name)}  return this.name+this.age;  }  person.prototype.new = "08976567";  person.prototype.age = function() {console.log("Hello my age is " + this.age) };  var p1= new person(("John", 36)  console.log (p1.x)  console.log (p1.name)  console.log (p1.age)  p1.myfunc()  person().hello()  person() | trait message {  public function msg() { echo "OOP is fun! ";}  }  class Person {  use message;  public $name, $age;  public function \_\_construct($name,$age) {  $this->name = $name;  $this->age = $age;  self::welcome();  }  function \_\_destruct() {  echo "The class is over {$this->name}";  }  public function \_\_invoke(...$arguments){  echo 'Called to the \_\_invoke method';  }  public static function welcome() {  echo "Hello World!";  }  public $x=5;  protected $y;  private $z;  static $sch="new school";  const MSG = "Thank you”;  static function hello() { echo "Hello!!"; }  function myfunc() {  echo "Hello my name is ".$this->name;  }  //access any var,const,func with  self::$value|CONST|$func();  or $this->value|func()  }  final class Classname { } // use final on class or method to stop override  echo Person(); //welcome()  echo Person::MSG;  echo Person::$sch;  echo Person::hello();  $p1 = new Person("John", 36);  $p1() // \_\_invoke  $p1->msg();  $p1->myfunc();  echo $p1->name;  echo $p1->age;  echo $p1->x;  $p1->age=30;  $p1->x=34;  Class TestClass{  }  namespace User  use Html as H;  class Person{ … }  $p1 = new User/Person()  $p1 = new H() |
| Class Inheritance | class Student(Person): //inherits Person  def \_\_init\_\_(this, name, age, class):  super().\_\_init\_\_(name, age)  this.class=class  def welcome(this):  print("This " + this.name + " "+ this.age, ":the class of", this.class )  x = Student("Mike", 40, 2019) | class Student extends Person{ //inherits Person  constructor(name, age, class){  super(name,age)  this.class=class  }  welcome(){  console.log("This " + this.name + " "+ this.age,  ":the class of", this.class)  }  }  var x = new Student("Mike", 40, 2019) | interface School {  public function address();  }  class Student extends Person implements School{  function \_\_construct($class){  $this->class=$class;  Parent::welcome()  }  public function address() {  echo “somewhere";  }  function welcome(){  echo "This " + $this->name + " "+ $this->age,  ":the class of", $this->class;  }  }  $x = new Student("Mike", 40, 2019); |
| Error Handling | try:  except:  except NameError:  else:  finally: | try{ throw ‘error’}  catch(err){ err.name, err.messgae }  finally{} | try{ throw new Exception(‘’) }  catch(Exception $e){ $e->getMessage(); } |
| Accept CLI Arg |  |  |  |